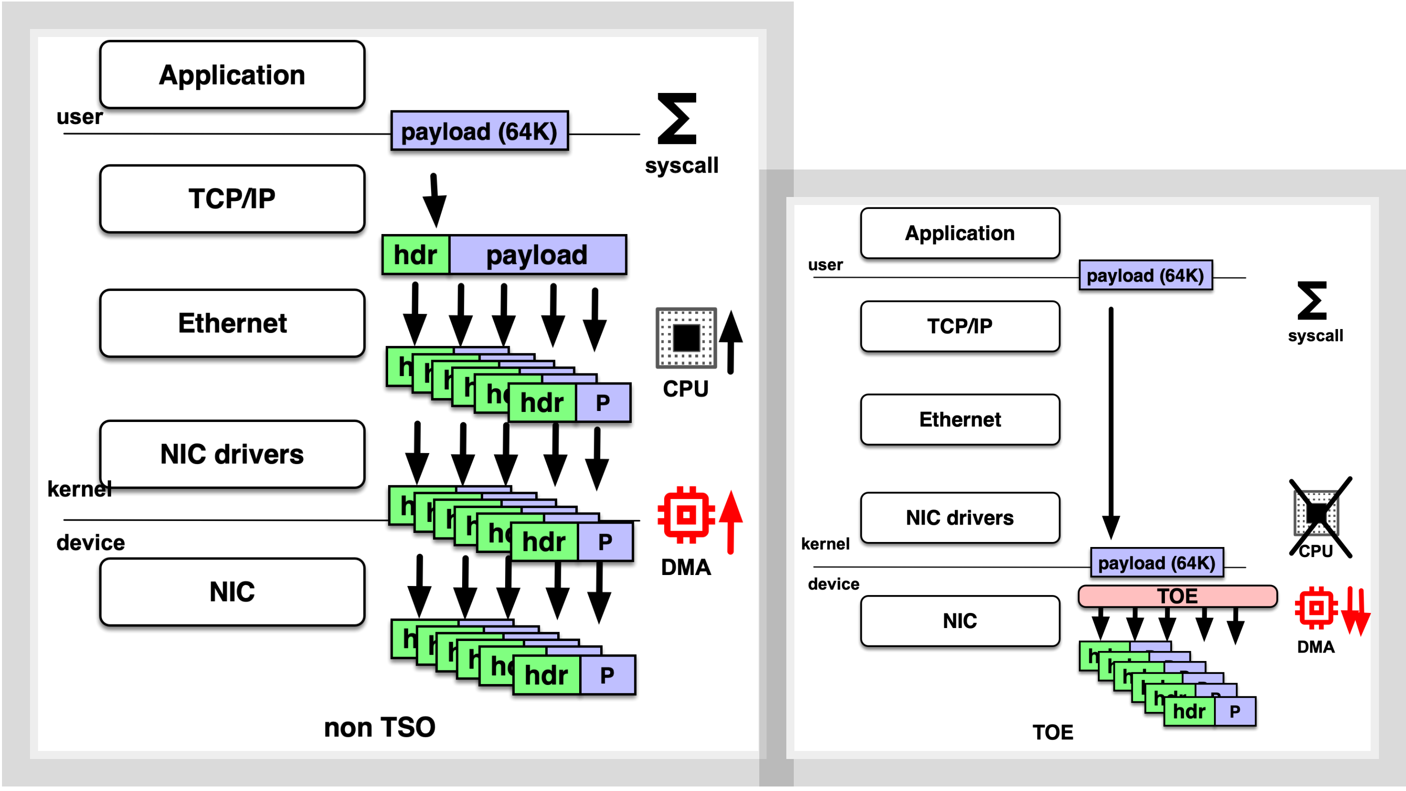
# kernel offload with complete host kernel functionalities

Ryo Nakamura (u-tokyo), **Hajime Tazaki (iijlab)**

Linux netdev conference 0x17 (2023)

# TCP Offload Engine (ToE)

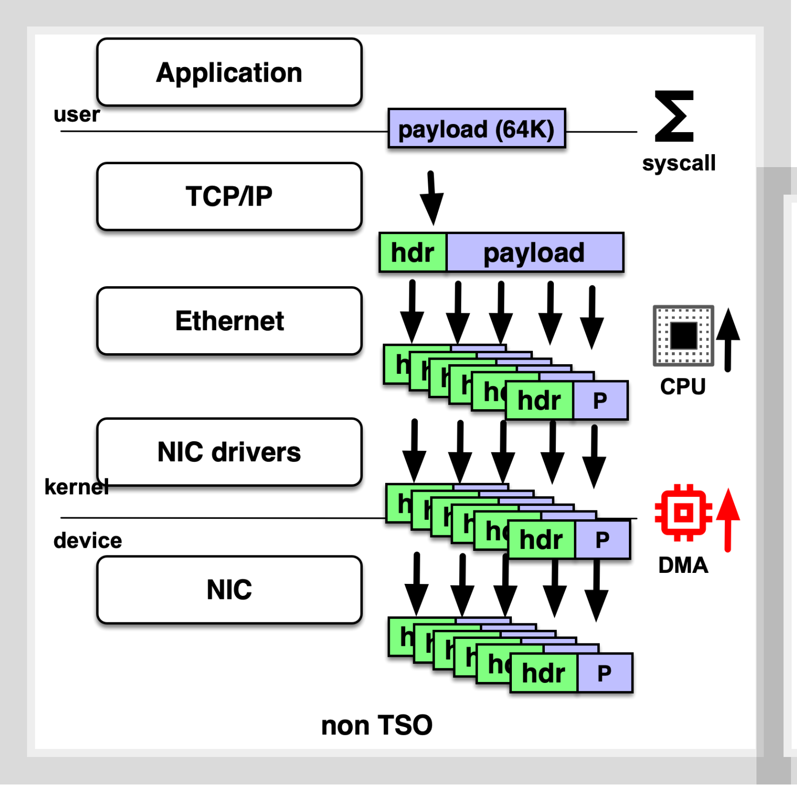
- So, (fully) offload TCP to NIC
  - save CPU cycles (TCP protocol handles at NIC, not host)
  - save DMA (ACKs are from NIC, not host)
  - if data is also on NIC, app=>NIC copies are also offloaded
- **Do heavy-lifting on hardware**

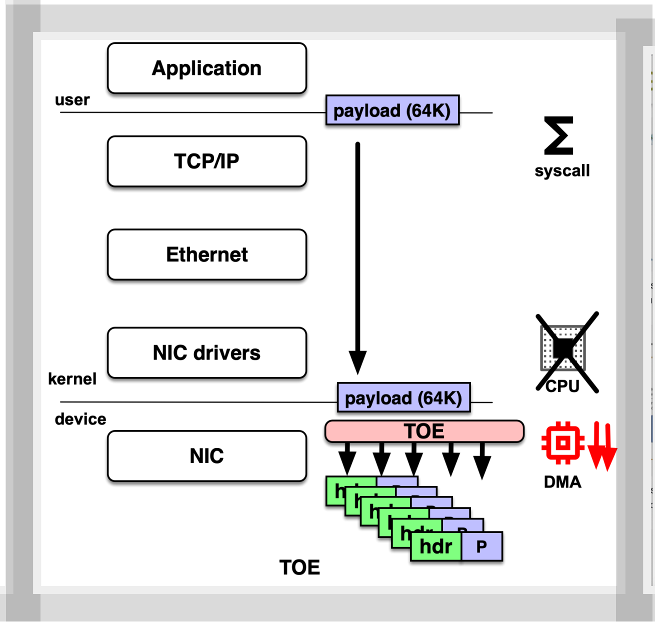


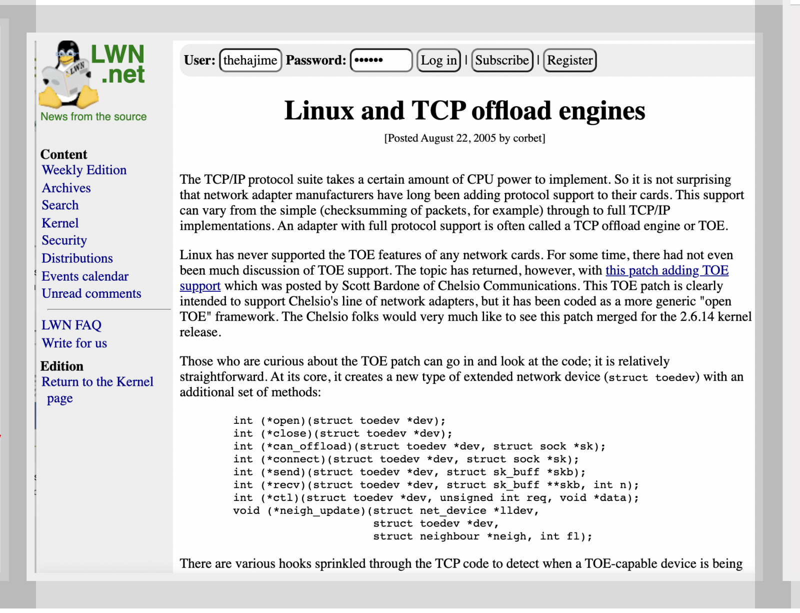
https://lwn.net/Articles/148697/
https://techcommunity.microsoft.com/t5/core-infrastructure-and-security/why-are-we-deprecating-network-performance-features-kb4014193/ba-p/259053

# TCP Offload Engine (ToE)

- So, (fully) offload TCP to NIC
  - save CPU cycles (TCP protocol handles at NIC, not host)
  - save DMA (ACKs are from NIC, not host)
  - if data is also on NIC, app=>NIC copies are also offloaded
- **Do heavy-lifting on hardware**
- but ToE been un-recommended
  - Linux **never** accepts TCP offload engines patch (2005)
  - Deprecation of Microsoft Chimney (2017)







## Linux and TCP offload engines

[Posted August 22, 2005 by corbet]

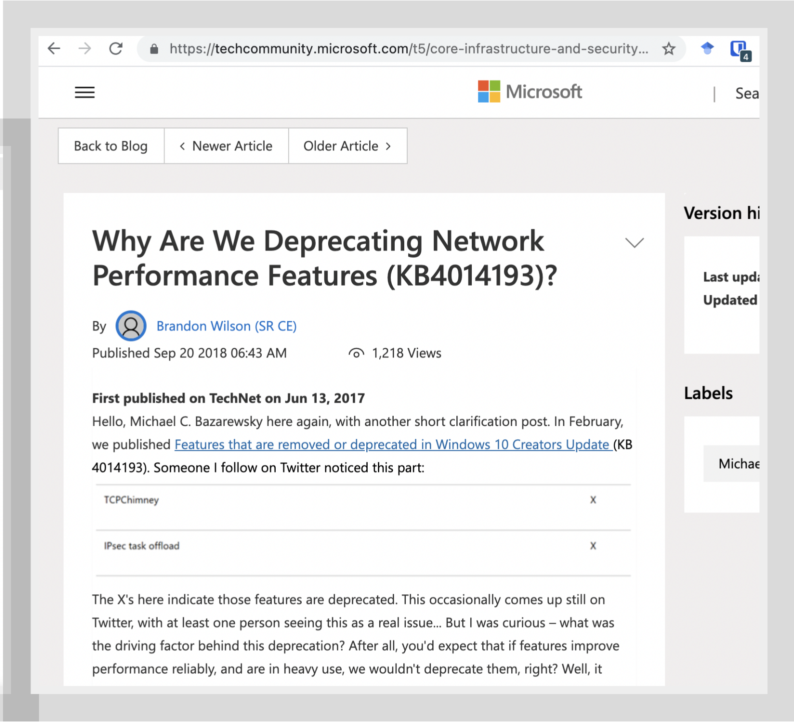
The TCP/IP protocol suite takes a certain amount of CPU power to implement. So it is not surprising that network adapter manufacturers have long been adding protocol support to their cards. This support can vary from the simple (checksumming of packets, for example) through to full TCP/IP implementations. An adapter with full protocol support is often called a TCP offload engine or TOE.

Linux has never supported the TOE features of any network cards. For some time, there had not even been much discussion of TOE support. The topic has returned, however, with this patch adding TOE support which was posted by Scott Bardone of Chelsio Communications. This TOE patch is clearly intended to support Chelsio's line of network adapters, but it has been coded as a more generic "open TOE" framework. The Chelsio folks would very much like to see this patch merged for the 2.6.14 kernel release.

Those who are curious about the TOE patch can go in and look at the code; it is relatively straightforward. At its core, it creates a new type of extended network device (`struct toedev`) with an additional set of methods:

```
int (*open)(struct toedev *dev);
int (*close)(struct toedev *dev);
int (*can_offload)(struct toedev *dev, struct sock *sk);
int (*connect)(struct toedev *dev, struct sock *sk);
int (*send)(struct toedev *dev, struct sk_buff *skb);
int (*recv)(struct toedev *dev, struct sk_buff **skb, int n);
int (*ctl)(struct toedev *dev, unsigned int req, void *data);
void (*neigh_update)(struct net_device *lldev,
                     struct toedev *dev,
                     struct neighbour *neigh, int fl);
```

There are various hooks sprinkled through the TCP code to detect when a TOE-capable device is being



## Why Are We Deprecating Network Performance Features (KB4014193)?

By Brandon Wilson (SR CE)

Published Sep 20 2018 06:43 AM 1,218 Views

**First published on TechNet on Jun 13, 2017**

Hello, Michael C. Bazarewsky here again, with another short clarification post. In February, we published Features that are removed or deprecated in Windows 10 Creators Update (KB 4014193). Someone I follow on Twitter noticed this part:

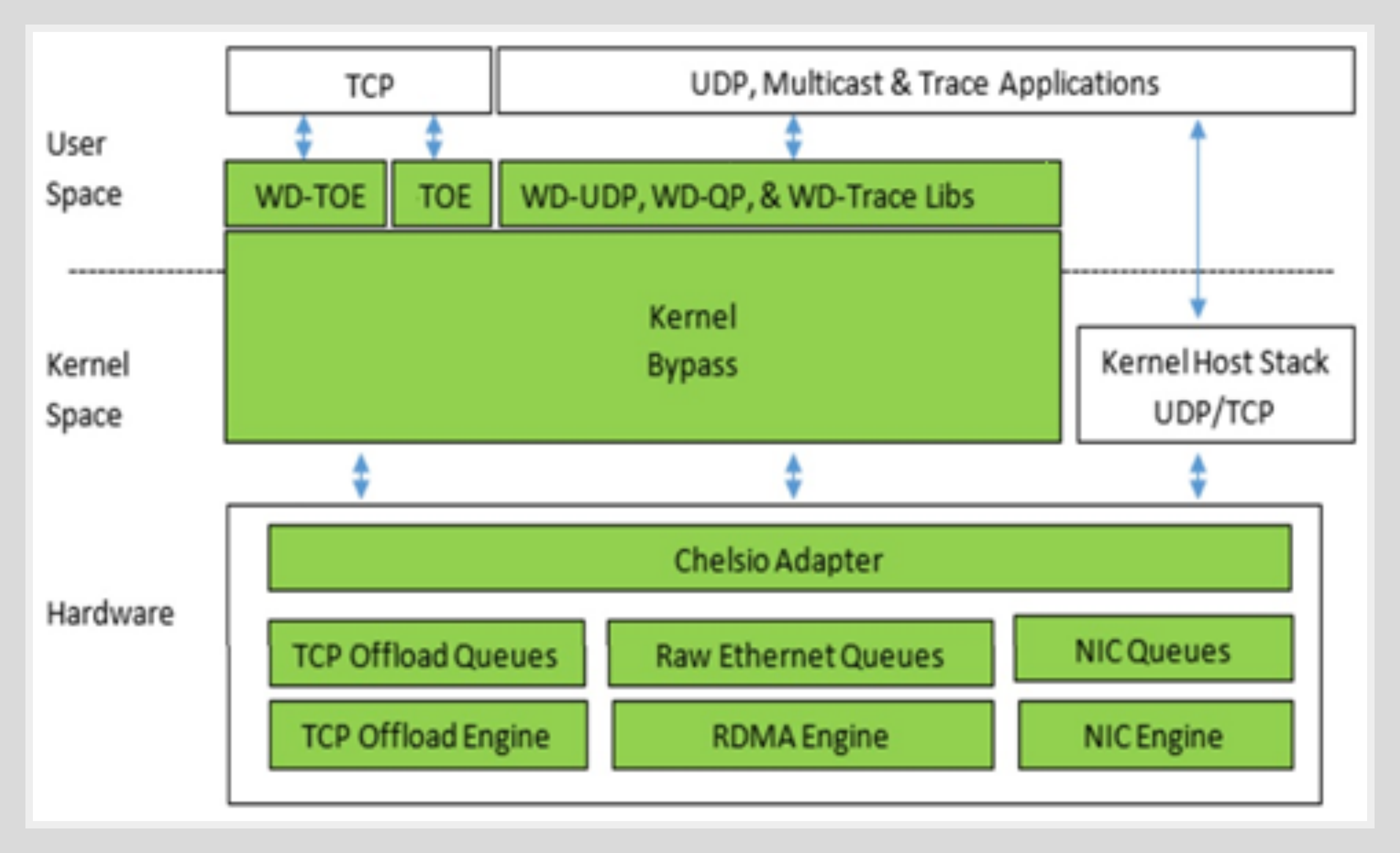
| | |
|---|---|
| TCPChimney | X |
| IPsec task offload | X |

The X's here indicate those features are deprecated. This occasionally comes up still on Twitter, with at least one person seeing this as a real issue... But I was curious – what was the driving factor behind this deprecation? After all, you'd expect that if features improve performance reliably, and are in heavy use, we wouldn't deprecate them, right? Well, it

https://lwn.net/Articles/148697/
https://techcommunity.microsoft.com/t5/core-infrastructure-and-security/why-are-we-deprecating-network-performance-features-kb4014193/ba-p/259053

# attempt to upstream ToE in Linux

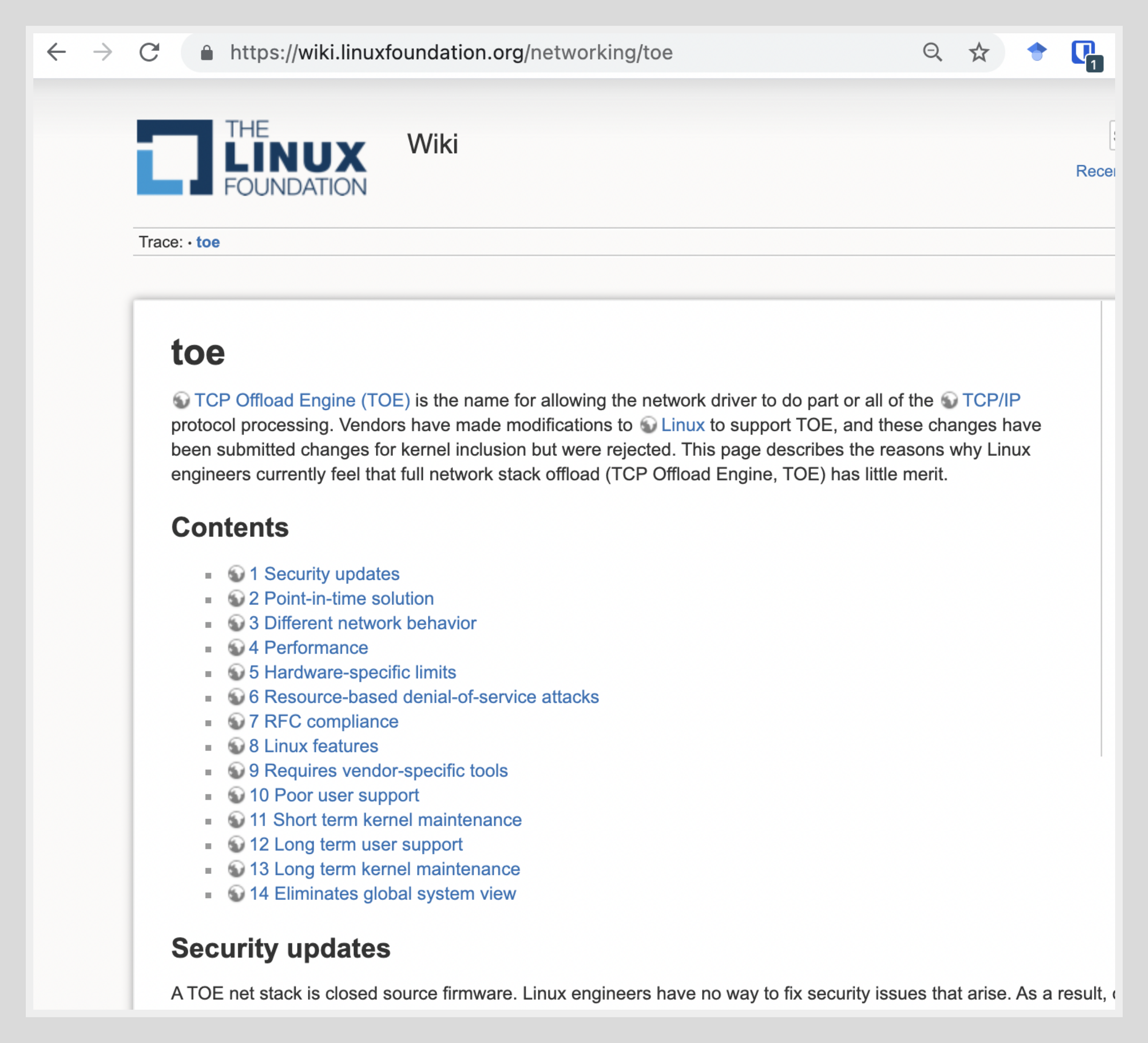
- 2005, Chelsio patch (ToE)
  - Abstract framework for various (vendor-specific) ToE NICs
  - Use Chelsio-TCP within Linux OS
- Reactions
  - security issue may not be fixed easily
  - Linux features are not involved: e.g., netfilter is skipped.



https://www.chelsio.com/wp-content/themes/chelsio/images/fsi_fig1.png

# why ToE was rejected ?

1. Security updates
2. Point-in-time solution
3. Different network behavior
4. Performance
5. Hardware-specific limits
6. Resource-based denial-of-service attacks
7. RFC compliance
8. Linux features
9. Requires vendor-specific tools
10. Poor user support
11. Short term kernel maintenance
12. Long term user support
13. Long term kernel maintenance
14. Eliminates global system view



Trace: · toe

## toe

TCP Offload Engine (TOE) is the name for allowing the network driver to do part or all of the TCP/IP protocol processing. Vendors have made modifications to Linux to support TOE, and these changes have been submitted changes for kernel inclusion but were rejected. This page describes the reasons why Linux engineers currently feel that full network stack offload (TCP Offload Engine, TOE) has little merit.

### Contents

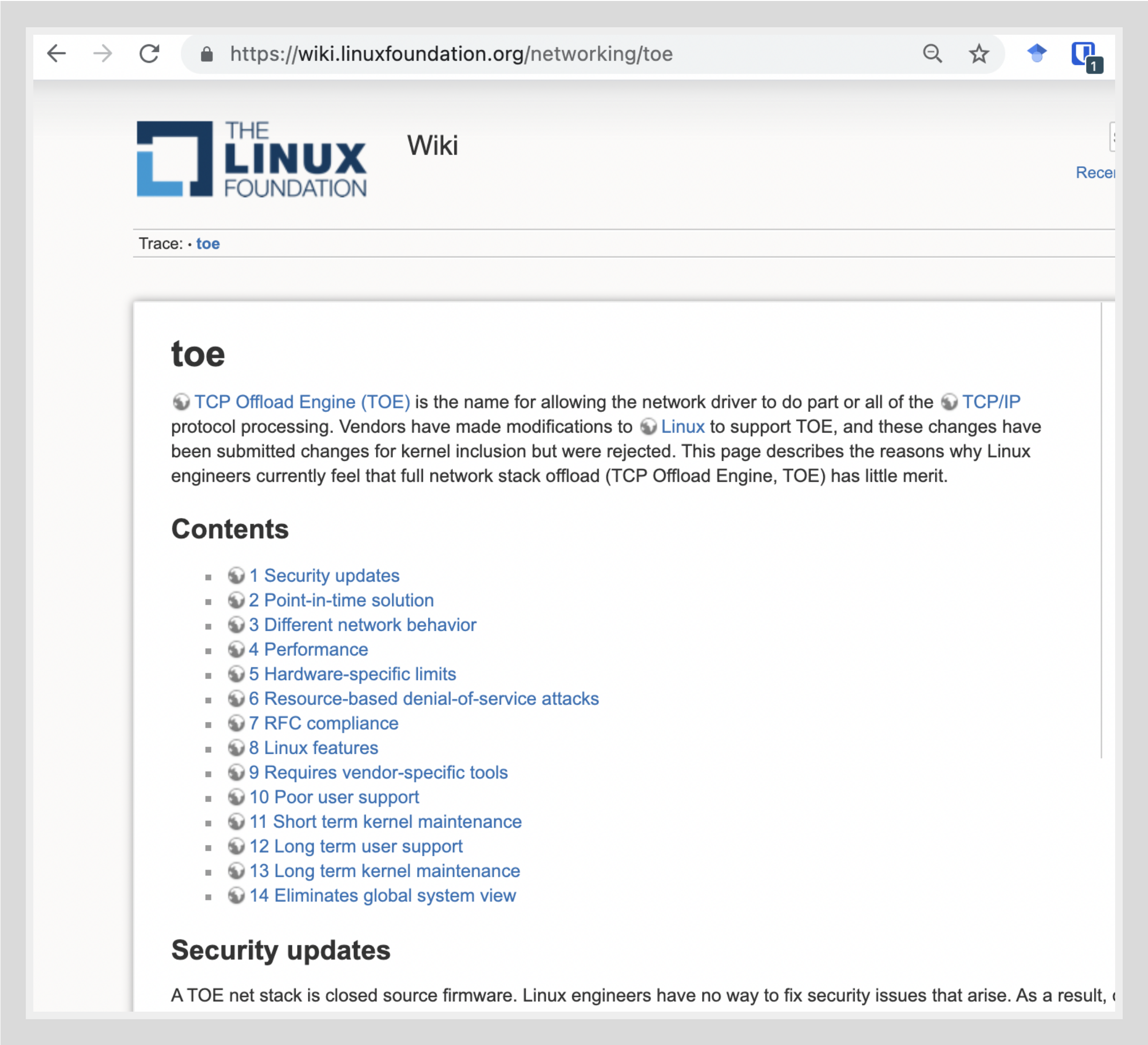
### Security updates

A TOE net stack is closed source firmware. Linux engineers have no way to fix security issues that arise. As a result,

https://wiki.linuxfoundation.org/networking/toe

# reasons of ToE rejected

- **Lack of featureset** (no netfilter on ToE)
  - 3.Different network behavior
  - 7.RFC compliance
  - 8.Linux features
- **Lack of governance** (cannot control from kernel developer)
  - 1. Security updates
  - 12.Long term user support
  - 14.Eliminates global system view
- **Different ecosystem** (lifetime: decades <=> few years)
  - 11.Short term kernel maintenance
  - 12.Long term user support
  - 13.Long term kernel maintenance
- **Different TCP implementation** (vendor specific)
  - 8.Linux features
  - 9.Requires vendor-specific tools



https://wiki.linuxfoundation.org/networking/toe

THE LINUX FOUNDATION Wiki

Trace: · toe

## toe

TCP Offload Engine (TOE) is the name for allowing the network driver to do part or all of the TCP/IP protocol processing. Vendors have made modifications to Linux to support TOE, and these changes have been submitted changes for kernel inclusion but were rejected. This page describes the reasons why Linux engineers currently feel that full network stack offload (TCP Offload Engine, TOE) has little merit.

### Contents

- 1 Security updates
- 2 Point-in-time solution
- 3 Different network behavior
- 4 Performance
- 5 Hardware-specific limits
- 6 Resource-based denial-of-service attacks
- 7 RFC compliance
- 8 Linux features
- 9 Requires vendor-specific tools
- 10 Poor user support
- 11 Short term kernel maintenance
- 12 Long term user support
- 13 Long term kernel maintenance
- 14 Eliminates global system view

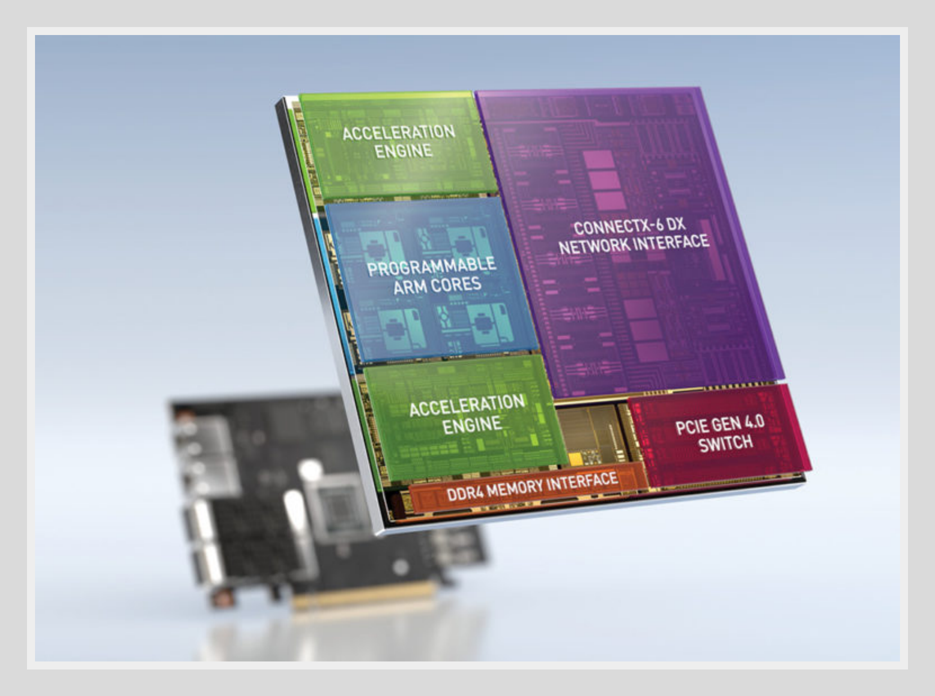
### Security updates

A TOE net stack is closed source firmware. Linux engineers have no way to fix security issues that arise. As a result,

https://wiki.linuxfoundation.org/networking/toe

# what can we do ?

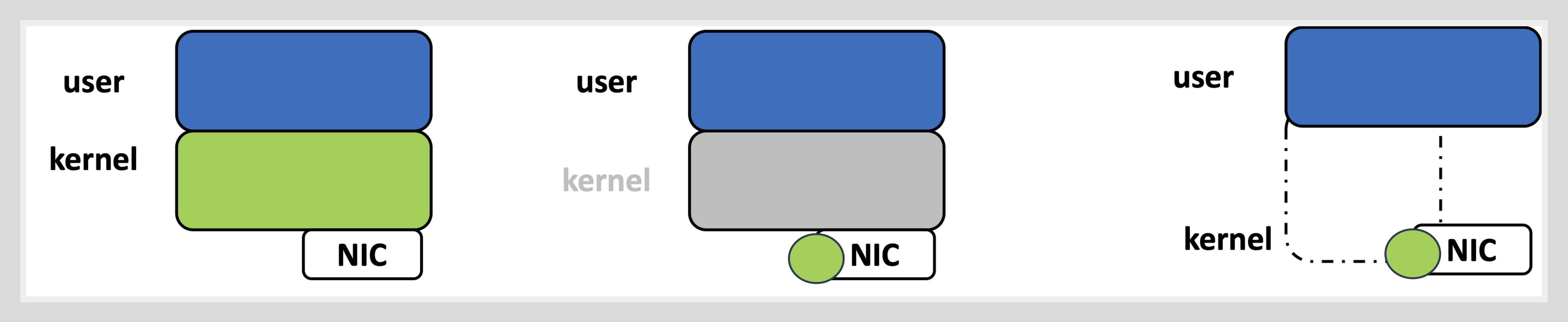
- it was around 2005, and it's 2023
- now NICs have Linux running inside (DPUs/SmartNICs)
- **worthwhile to try ToE ?**



https://www.hpcwire.com/2021/12/21/nvidia-touts-bluefield-2-performance-disputes-fungible-claim-are-dpu-wars-ahead/

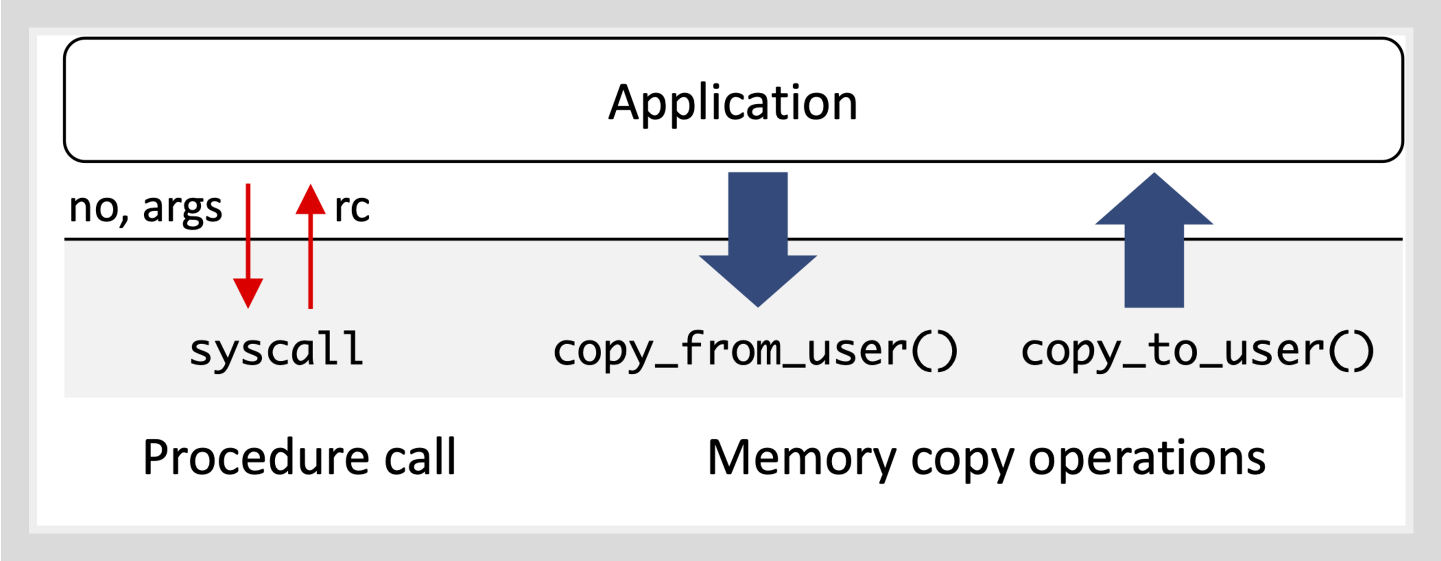
# our attempt: kernel offload

- **mino** (a random, tentative name)
- basic idea: **decouple kernel from host**
  - Split kernel functions to (Smart)NICs
  - Use user/kernel space memory abstraction via RDMA channel
  - Unified/Unchanged view from userspace applications
- Benefits
  - no drastic ABI change btw/ user/kernel spaces
  - existing tools compatible (iproute2, /proc, /sys files)
  - clean abstraction, plug-gable kernel implementation
  - still software-based; thus updatable

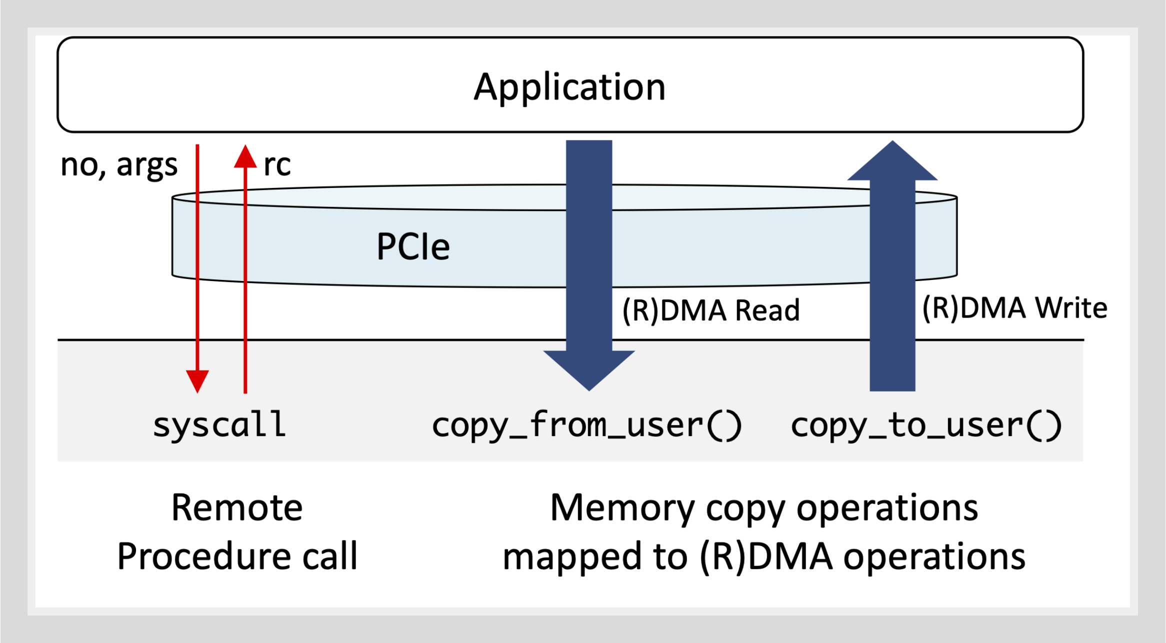


# internals

split kernel at ***copy_{from,to}_user()***



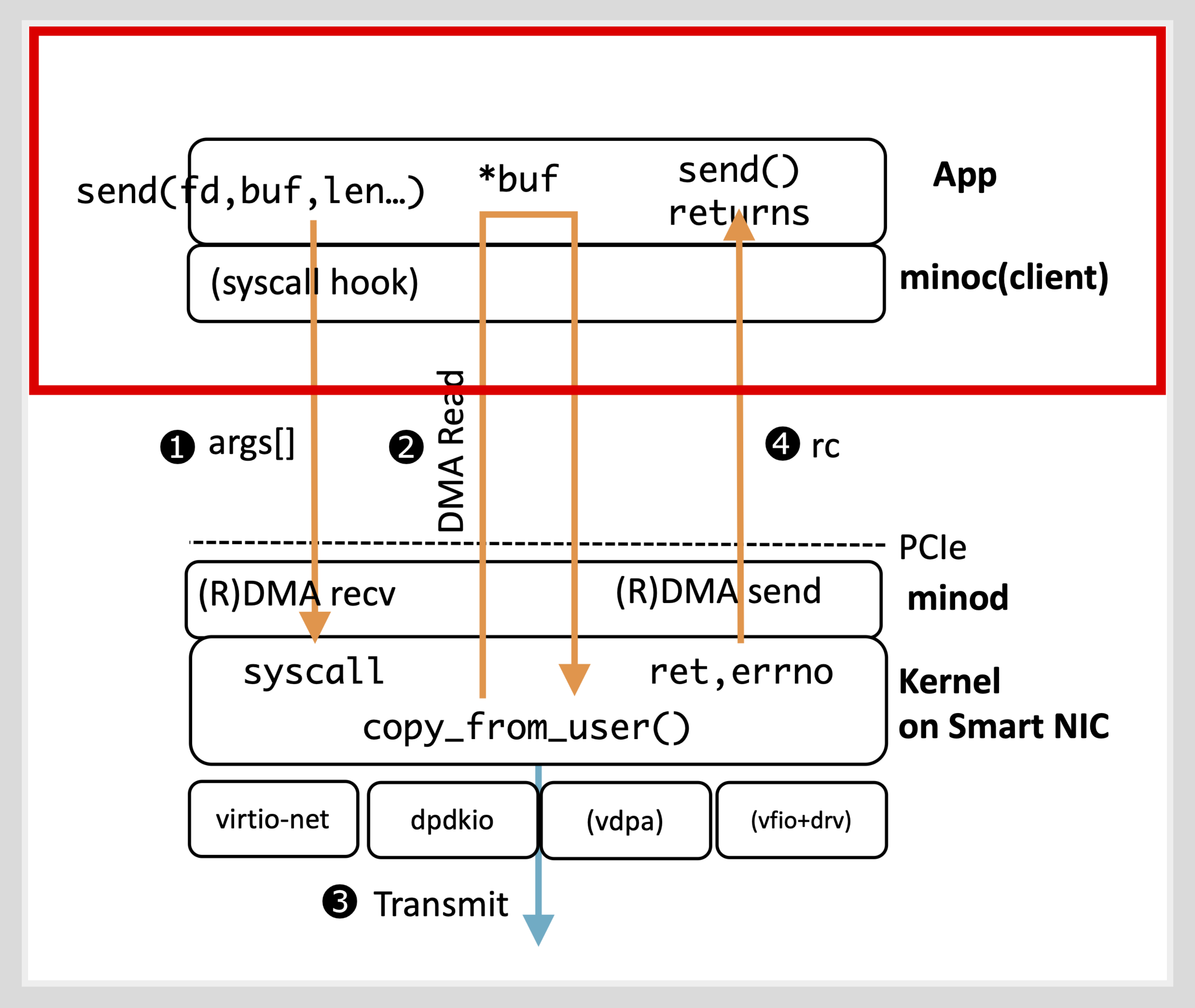
typical syscall



syscall w/ kernel offload

# Host part: minoc

- 1) hook syscall (LD_PRELOAD, zpoline*1)
- 2) copy syscall reg and buffers (*buf) to NIC (mrcc/(R)DMA)
  - register buffers for RDMA read
- 3) wait for result (rc, errno) from rdma_get_recv_comp(3)

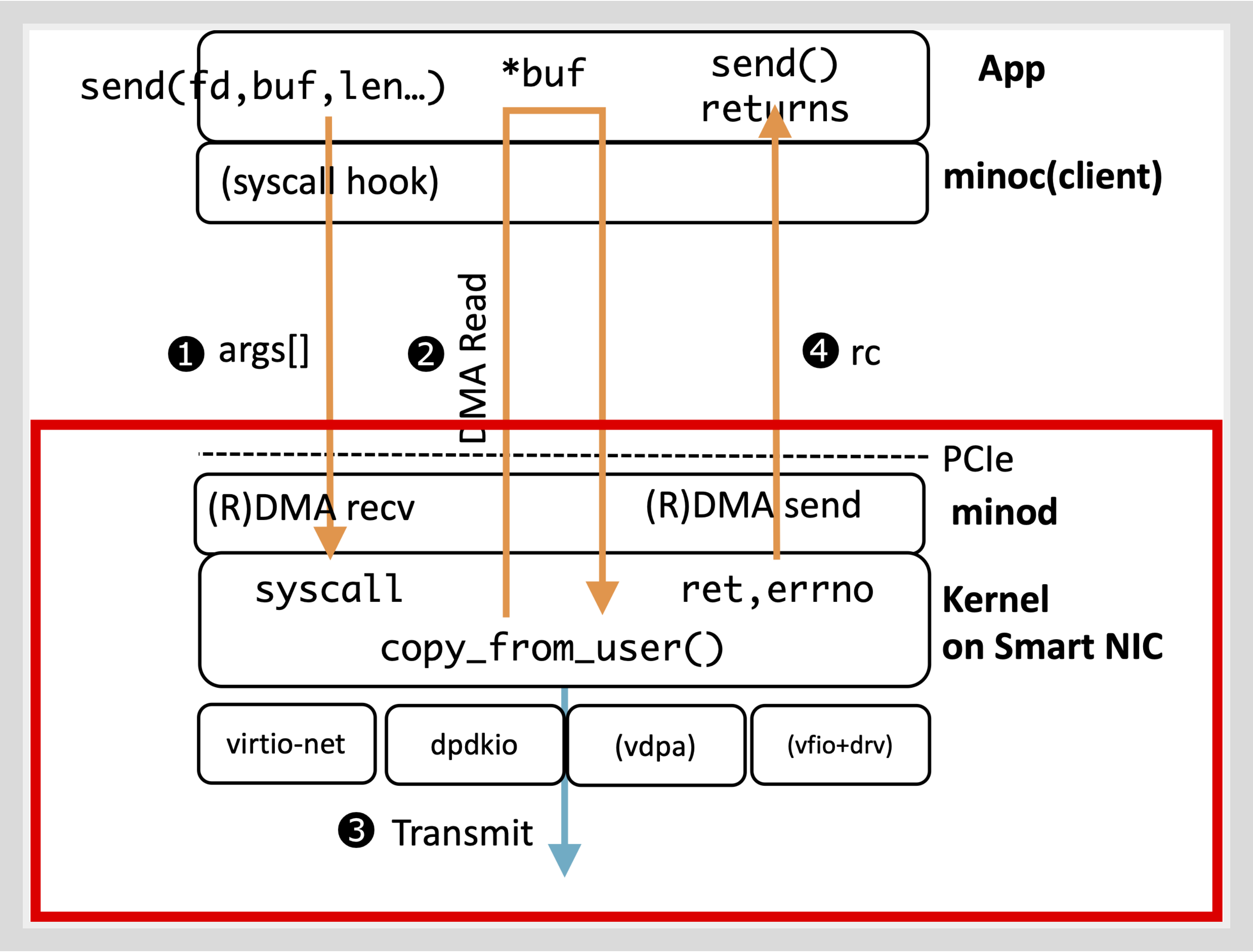


**run mino client at a host device**

*1 https://www.usenix.org/conference/atc23/presentation/yasukata

# NIC part: minod

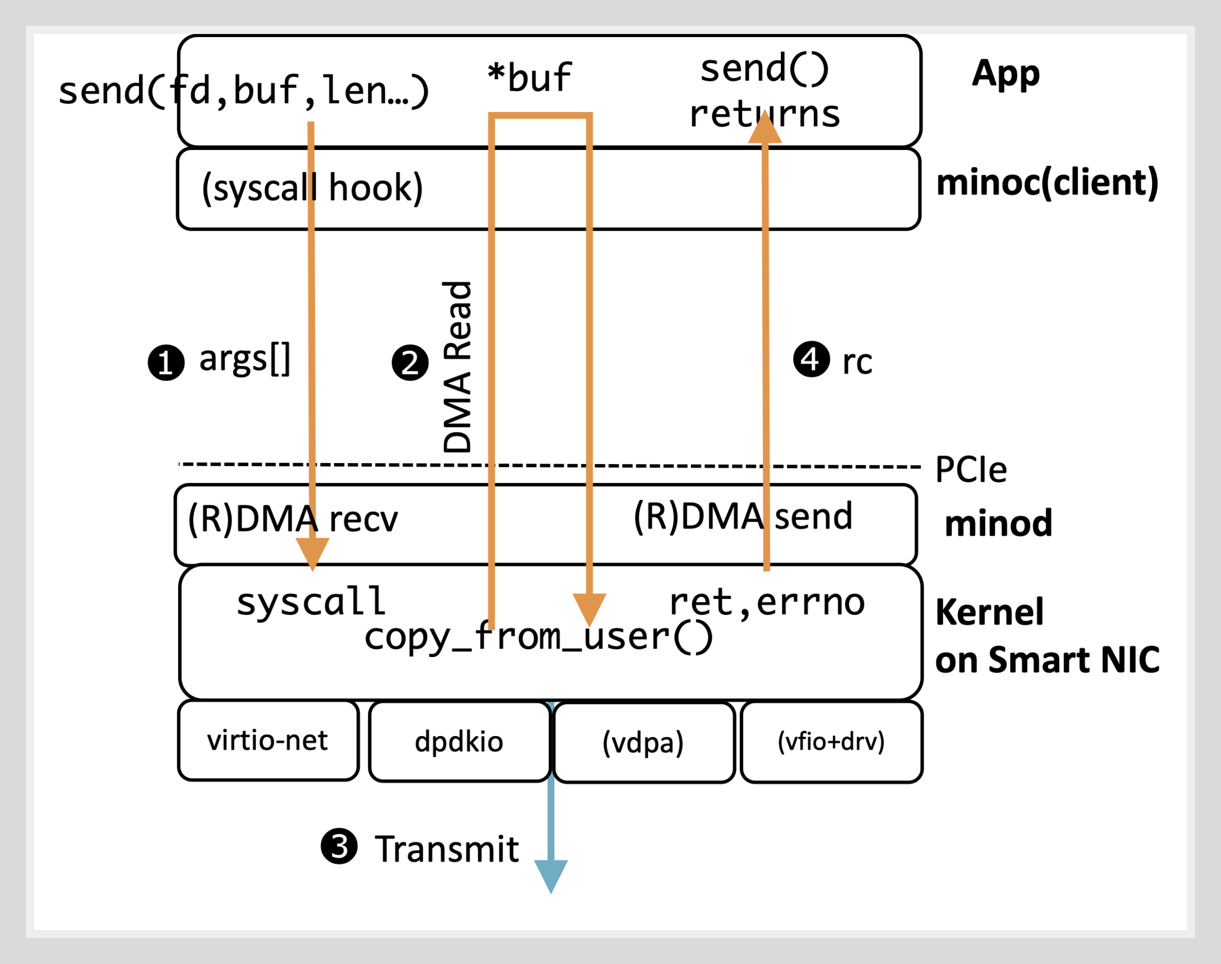
- 0) minod runs on userspace
  - minod **can** run on kernel space (LKM)
- 1) wait for a trigger via char dev (/dev/usrcall)
- 2) process syscall via `copy_from_user()`
- 3) (regular syscall handling)
- 4) post result to callee by `copy_to_user()`



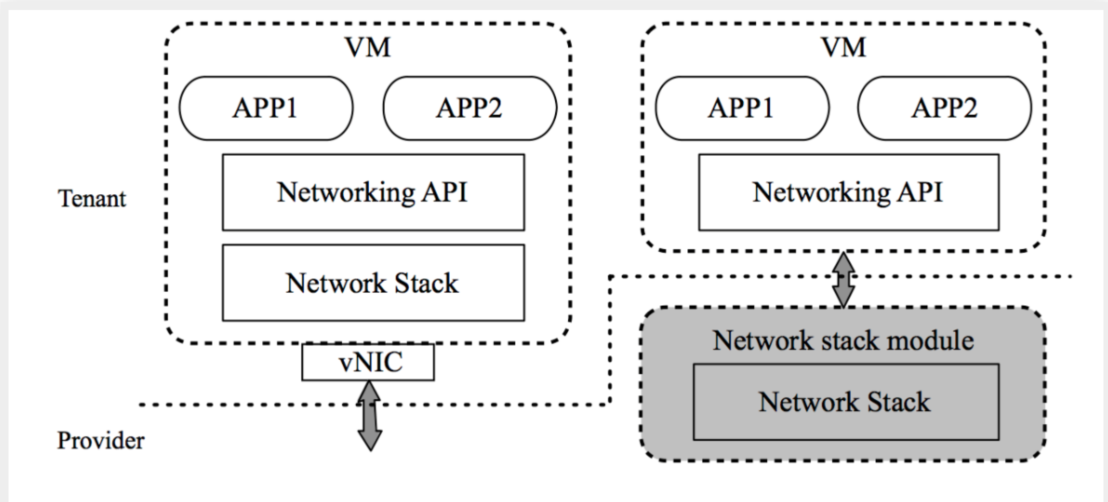
**run mino daemon process(es) at the offload device**

# multiple implementations of NIC side

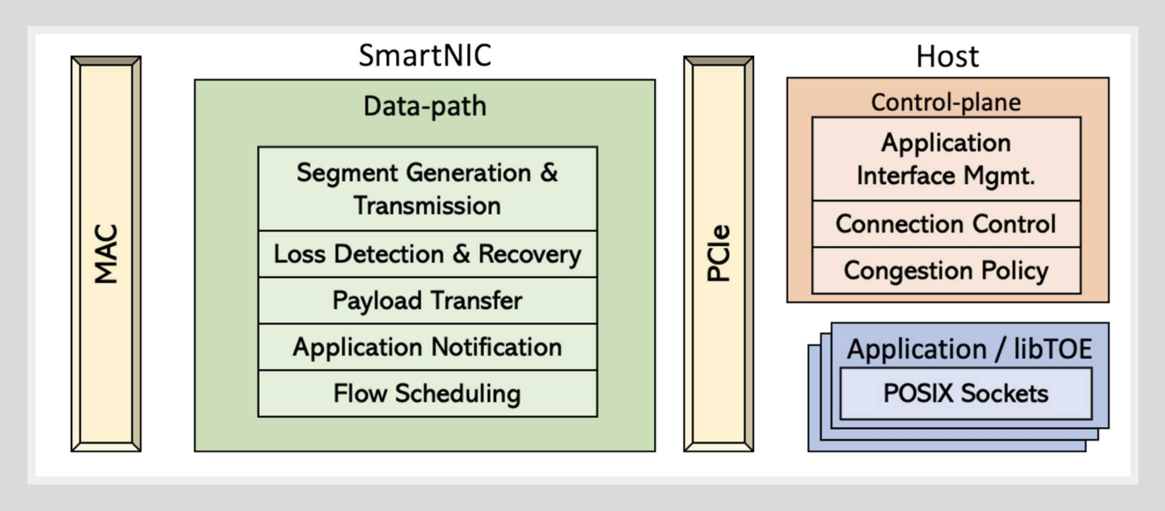
- A userspace process using LKL (Linux Kernel Library)
  - LKL exists to **reuse* Linux code in a different environment
- But not limited to use LKL
  - Can be implemented as a kernel module

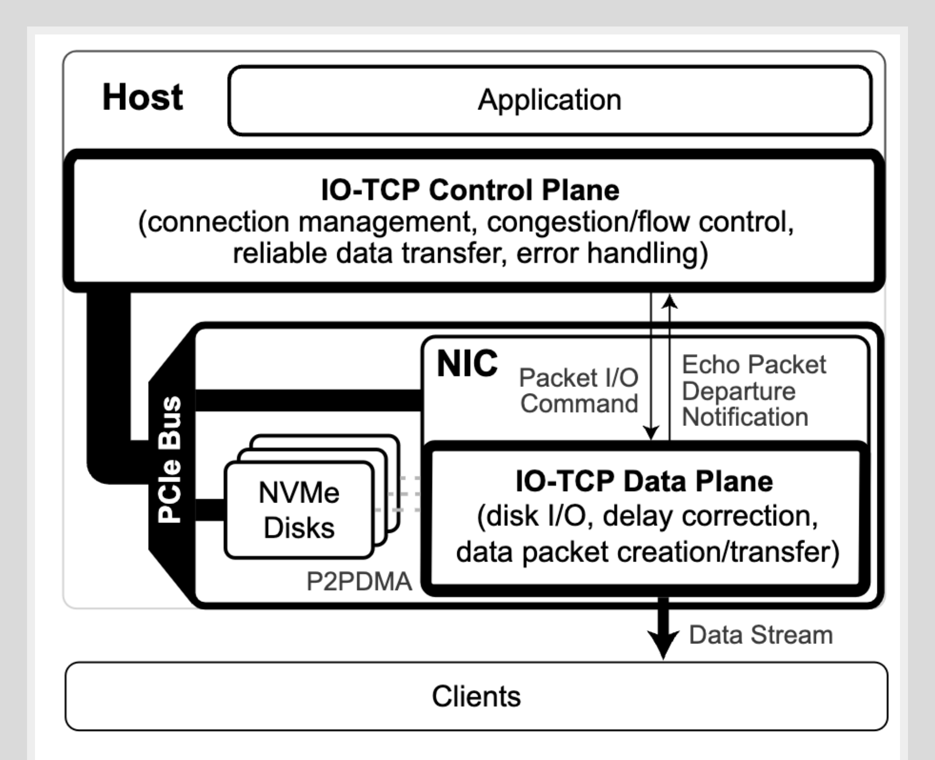


# alternatives



(a). Existing architecture (b). Network stack as a service




Host
Application
IO-TCP Control Plane
(connection management, congestion/flow control,
reliable data transfer, error handling)
PCIe Bus
NIC
Packet I/O
Command
Echo Packet
Departure
Notification
NVMe
Disks
P2PDMA
IO-TCP Data Plane
(disk I/O, delay correction,
data packet creation/transfer)
Data Stream
Clients

The overview of IO-TCP stacks

- Split kernels
  - netkernel (mTCP/Linux++ TCP impl.)
  - FlexTOE (TAS-based TCP impl.)
  - IO-TCP (mTCP-based TCP impl.)
- Chelsio T6
  - a classical ToE (own TCP impl.)

- Niu et al., NetKernel: Making Network Stack Part of the Virutalized Infrastructure, ATC '20
- Shashidhara et al., FlexTOE: Flexible TCP Offload with Fine-Grained Parallelism, NSDI '22
- Kim et al., Rearchitecting the TCP Stack for I/O-Offloaded Content Delivery, NSDI '23
- Terminator 6 ASIC https://www.chelsio.com/terminator-6-asic/

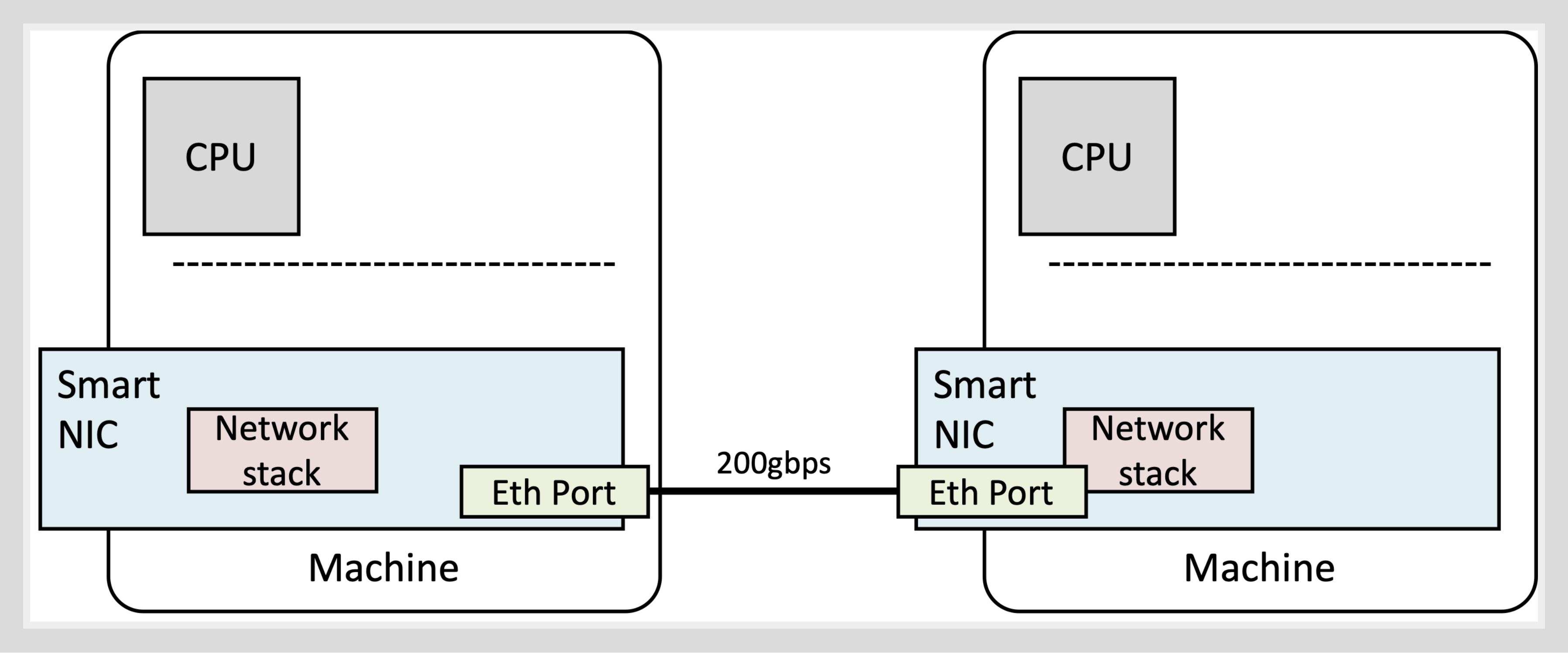
# alternatives

| | netkernel | flextoe | iotcp | chelsio | minod |
|---|---|---|---|---|---|
| 1. security update | 🙆 | 🙆 | 🙆 | 😢 | 🙆 |
| 2. point-in-time solution | 🙆 | 🙆 | 🙆 | 😢 | 🙆 |
| 3. different behavior | 😢/🙆 | 😢 | 😢 | 😢 | 🙆 |
| 4. performance | 🙆 | 🙆 | 🙆 | 🙆 | ❓ |
| 5. hardware-specific limits | 😢 | 😢 | 😢 | 😢 | 😢 |
| 6. DoS attacks | 😢/🙆 | 😢 | 😢 | 😢 | 🙆 |
| 7. RFC compliance | 😢/🙆 | 😢 | 😢 | 😢 | 🙆 |
| 8. linux features | 😢/🙆 | 😢 | 😢 | 😢 | 🙆 |
| 9. vendor-specific tools | 😢 | 😢 | 😢 | 😢 | 🙆 |
| 10. poor user support | 😢/🙆 | 😢 | 😢 | 😢 | 🙆 |
| 11. (short-term) maintenance | 🙆 | 🙆 | 🙆 | 😢 | 🙆 |
| 12. long-term support | 🙆 | 🙆 | 🙆 | 😢 | 🙆 |
| 13. (long-term) maintenance | 🙆 | 🙆 | 🙆 | 😢 | 🙆 |
| 14. global system view | 😢 | 😢 | 😢 | 😢 | 😢 |

https://wiki.linuxfoundation.org/networking/toe

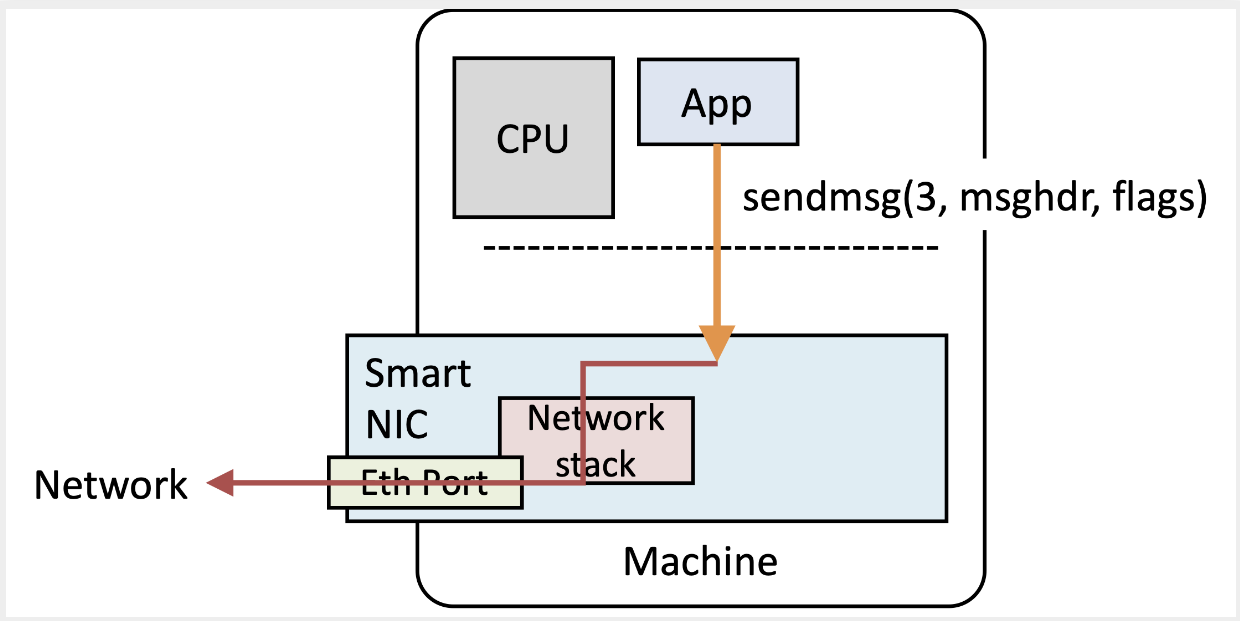
# Demo

# benchmark setup

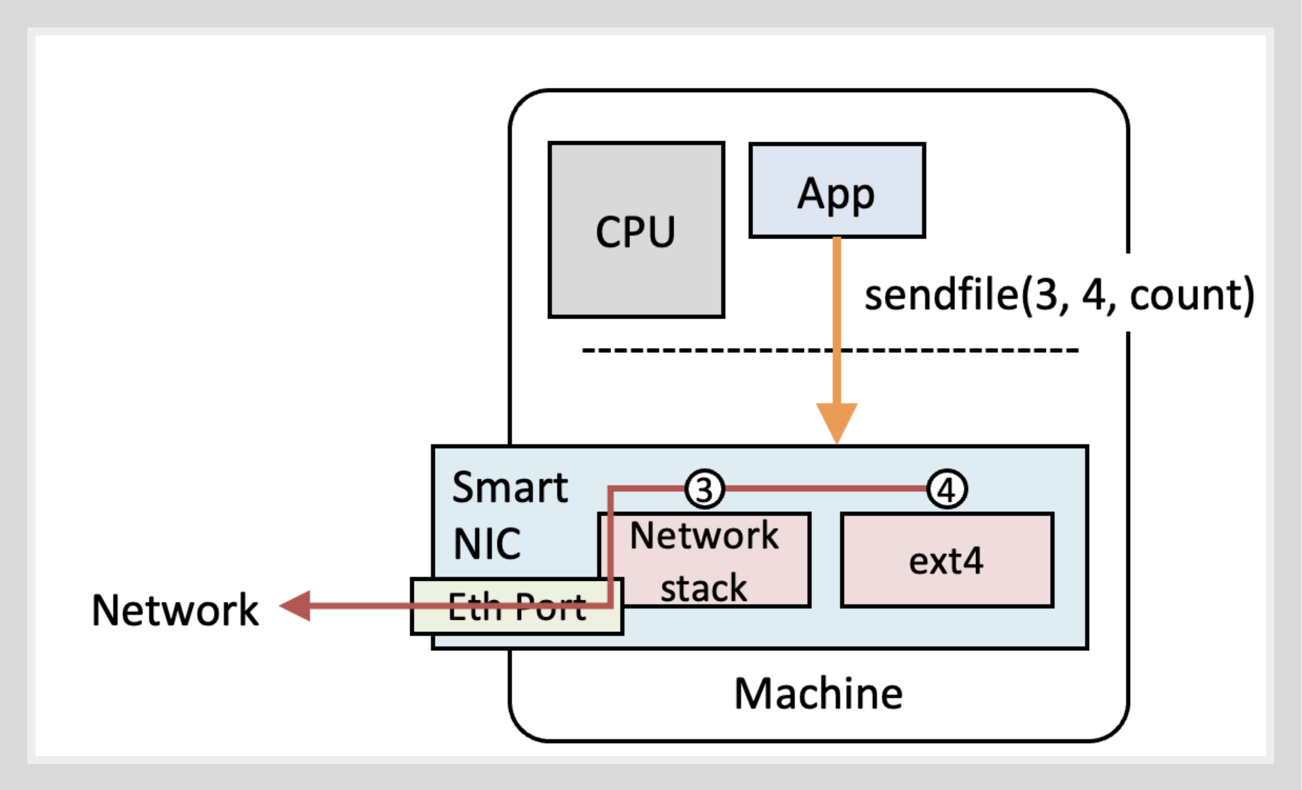


- 2 Machines (back-to-back)
- CPU: Xeon Gold 6326 CPU
- Bluefield-2 DPU: MBF2M345A-HECOT
  - x8 Armv8 A72 cores
  - 16GB RAM
  - 200G (QSFP56) 1port
- Workload
  - netperf TCP_STREAM
  - netperf TCP_SENDFILE
  - nginx + kTLS + SSL_sendfile
- Comparison
  - mino v.s. (host)Linux

# 1. netperf



sendmsg(2)

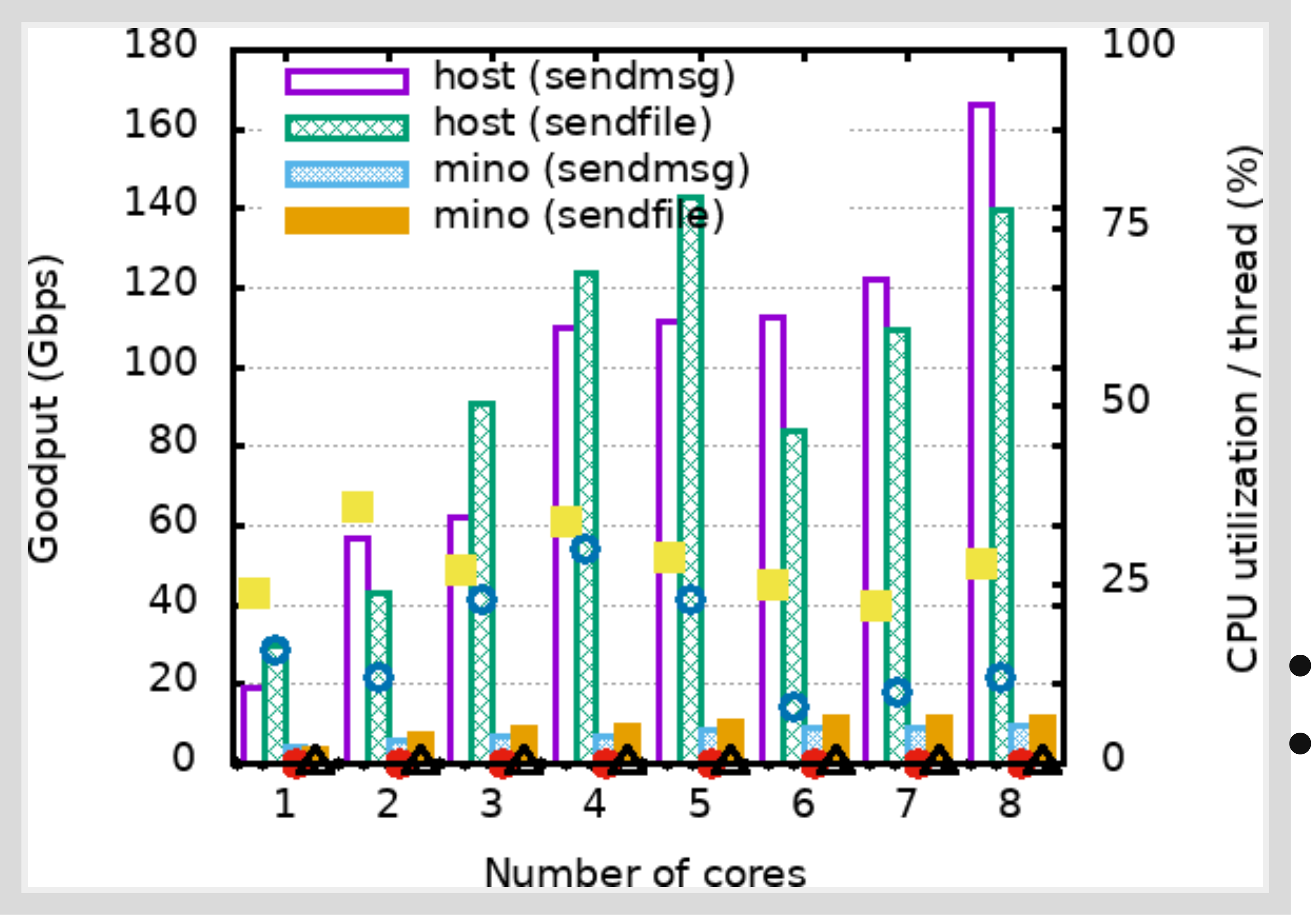


sendfile(2)

- sendfile should benefit a lot
- on NIC side: run multi-LKL instances
- measure cpu usage by `time` command

# 1. netperf (cont'd)

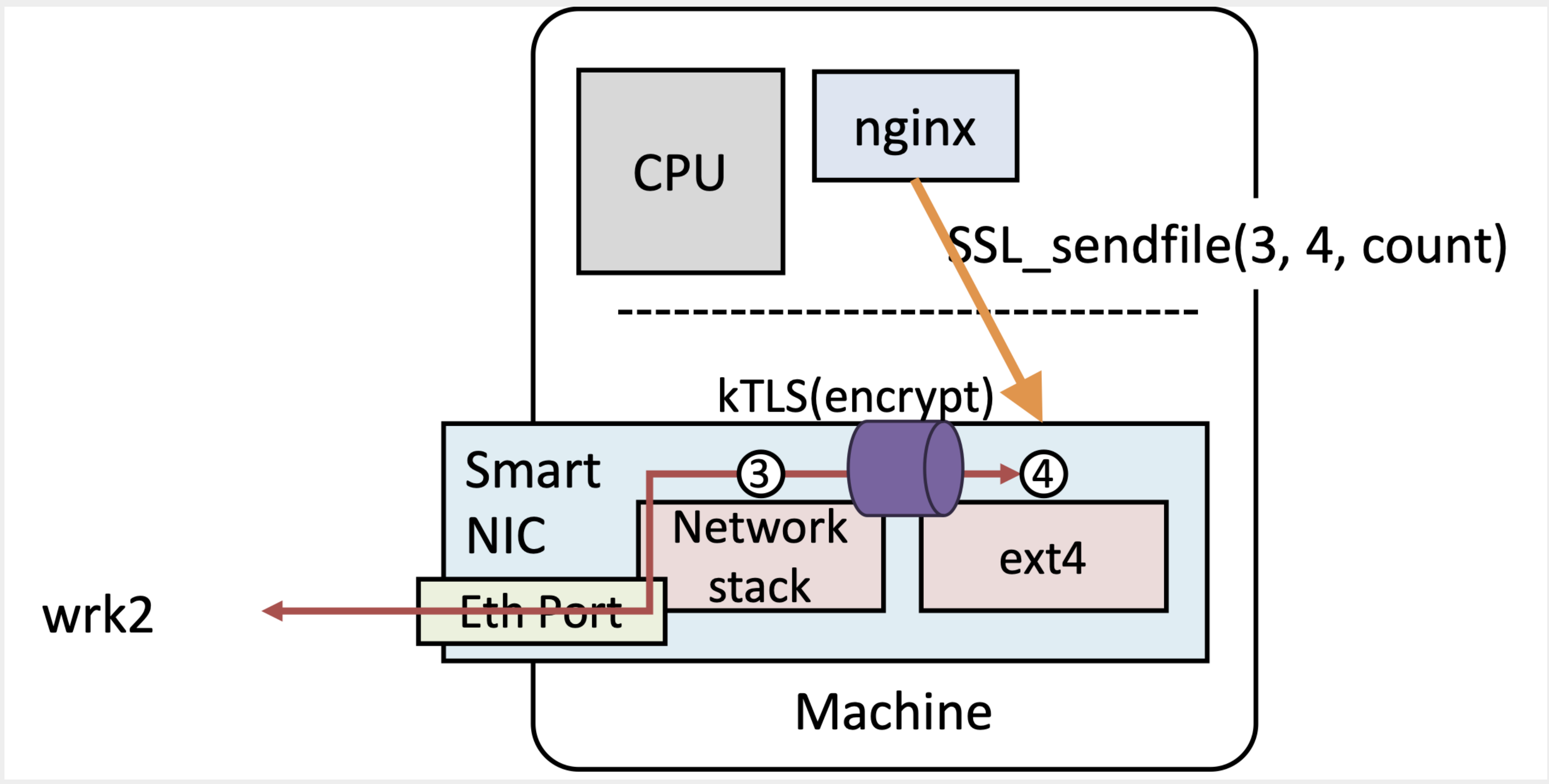
- goodput: always **host > mino** 😢
- cpu usage
  - mino: mostly zero
  - host: 20-40% (sendmsg), decrease a bit (sendfile)
- sendfile
  - does benefit on minod (kernel offload)
  - no stable benefit on host



- left-Y-axis: Throughput (Gbps)
- right-Y-axis: CPU usage (%)

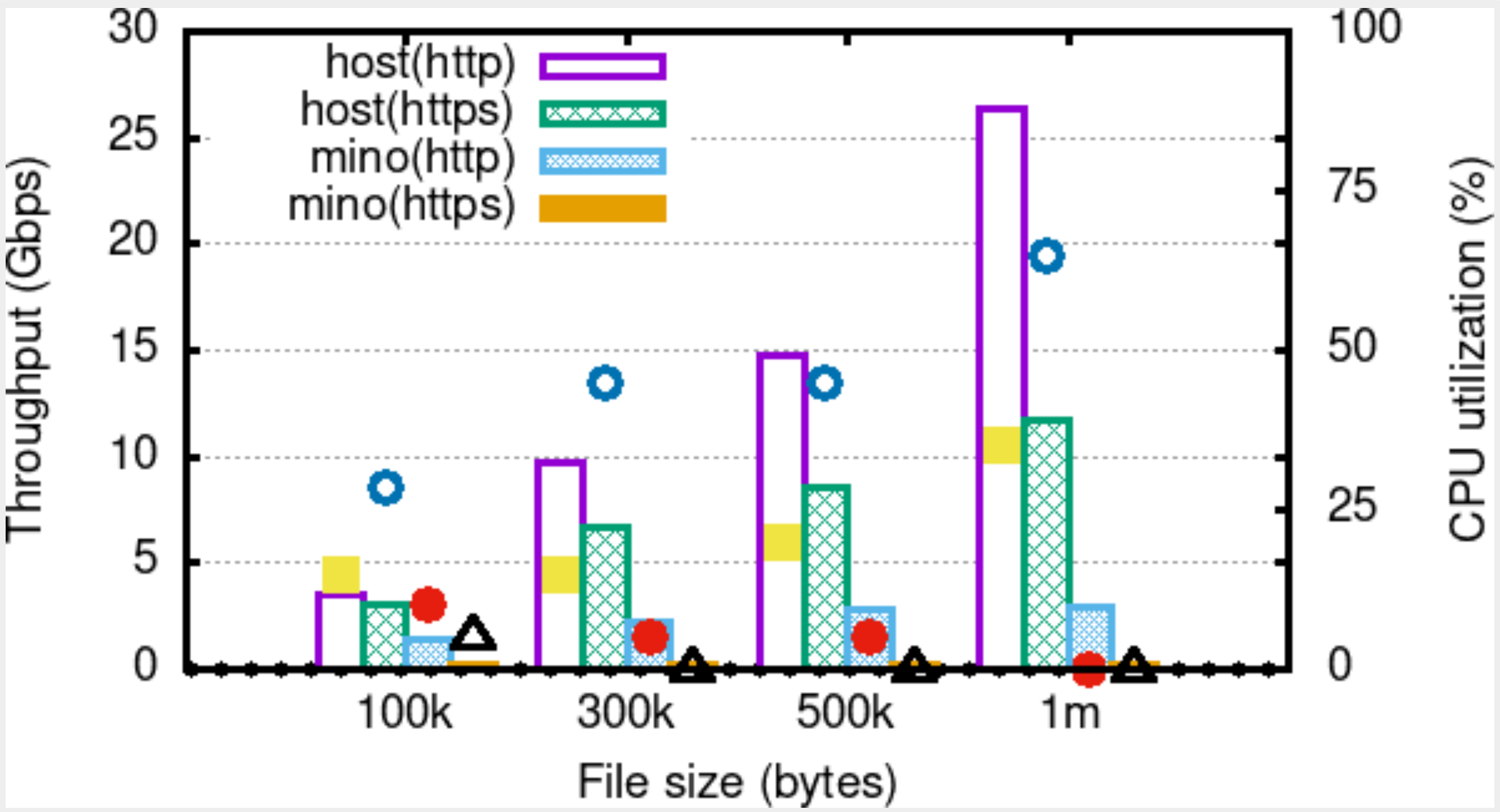
(num of core == num of parallel netperf processes)

# 2. nginx/wrk2, non-TLS/kTLS



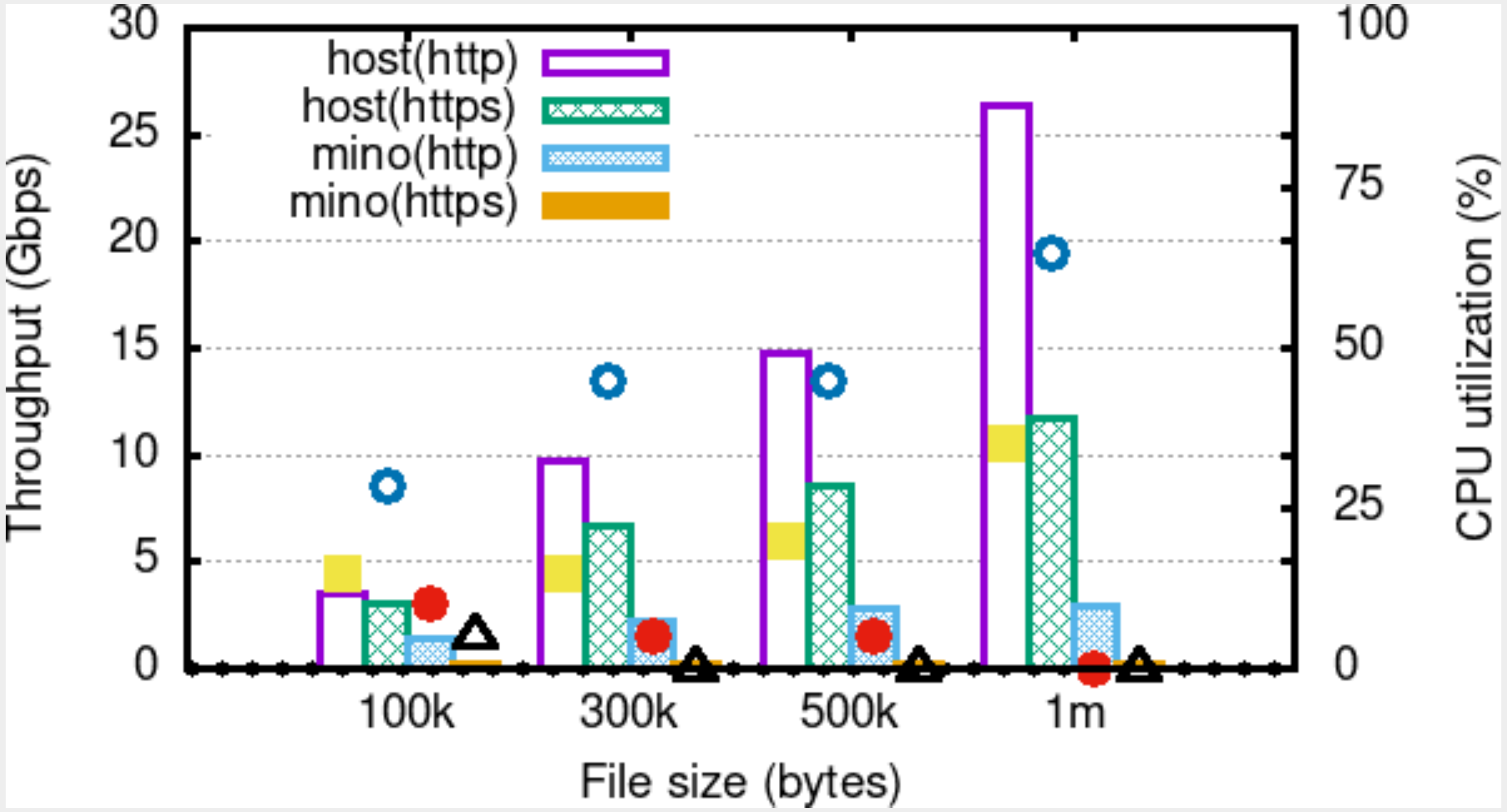
- nginx master (2023 Jun)
  - build w/ `--with-openssl-opt=enable-ktls`
  - nginx.conf `sendfile on;`
  - 1 worker process
  - openssl 3.0.9
- stressed with wrk2

# 2. nginx/wrk2, non-TLS/kTLS



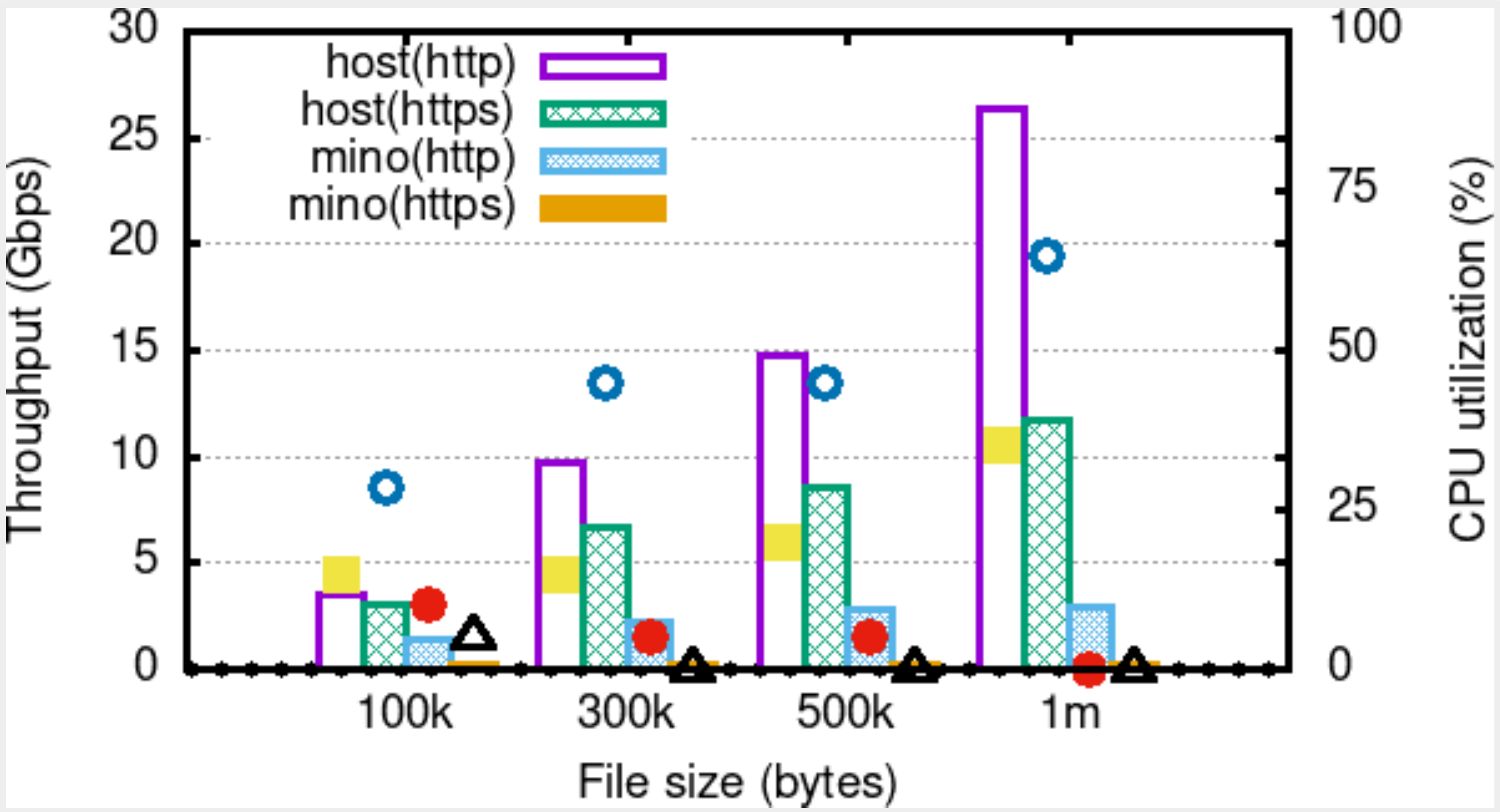
- left-Y-axis: Throughput (Gbps)
- right-Y-axis: CPU usage (%)

# 2. nginx/wrk2, non-TLS/kTLS



- left-Y-axis: Throughput (Gbps)
- right-Y-axis: CPU usage (%)

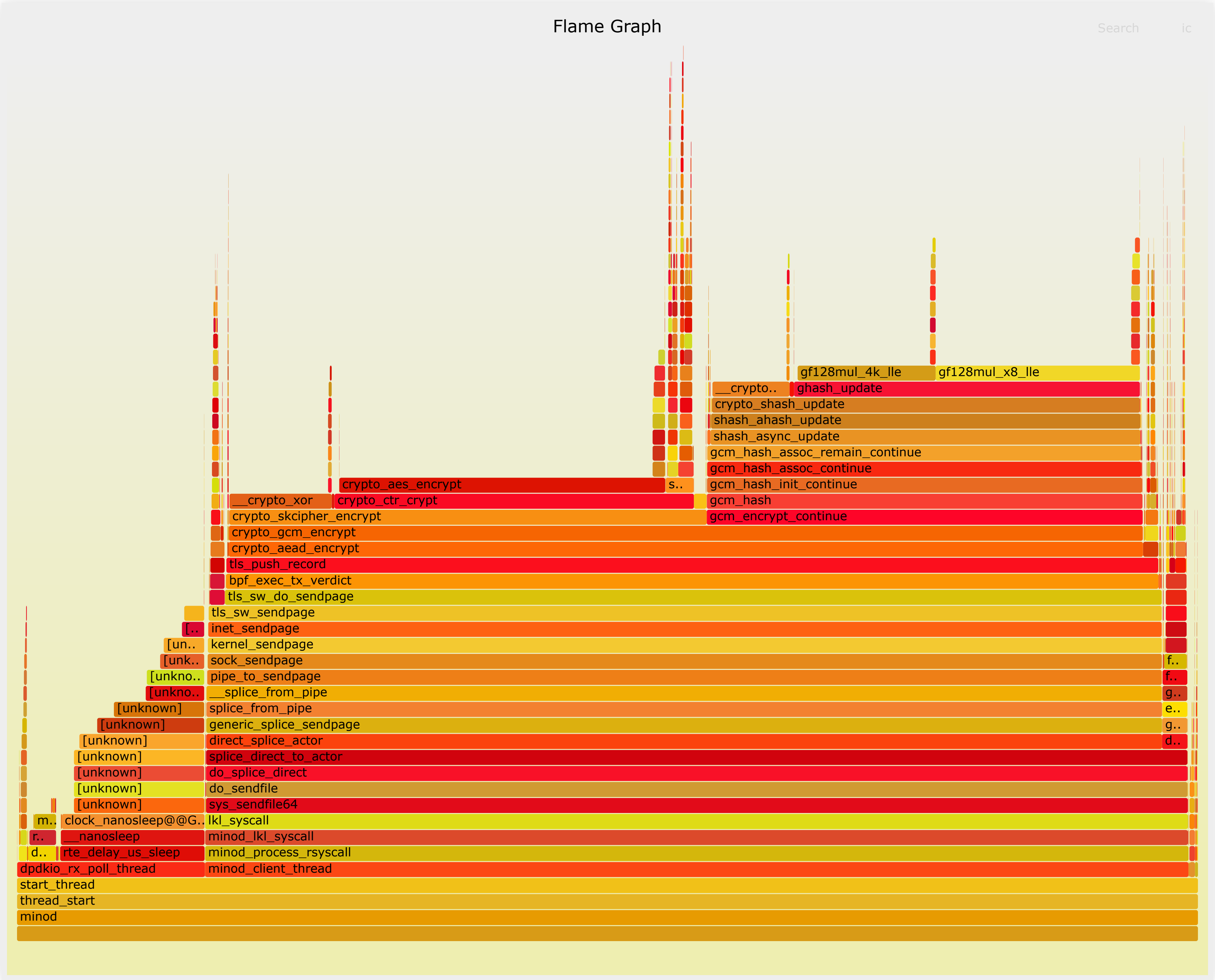
# 2. nginx/wrk2, non-TLS/kTLS



- left-Y-axis: Throughput (Gbps)
- right-Y-axis: CPU usage (%)

- goodput: always host > mino
- cpu usage
  - mino: almost zero (but less load..)
  - host: 20-40% (http), 25-70% bit (https)

# 2. nginw/wrk2 flamegraph (nginx)



(profiled on NIC (minod))

- spent 79.2% w/ `crypto_aead_encrypt`()
  - can be improved by crypto-offload

# What we saw ?

| | netkernel | flextoe | iotcp | chelsio | minod |
|---|---|---|---|---|---|
| security update | 🙆 | 🙆 | 🙆 | 😢 | 🙆 |
| point-in-time solution | 🙆 | 🙆 | 🙆 | 😢 | 🙆 |
| different behavior | 😢/🙆 | 😢 | 😢 | 😢 | 🙆 |
| performance | 🙆 | 🙆 | 🙆 | 🙆 | 😢 |
| hardware-specific limits | 😢 | 😢 | 😢 | 😢 | 😢 |
| DoS attacks | 😢/🙆 | 😢 | 😢 | 😢 | 🙆 |

# Observations

1. bottle neck: memory channel (host - NIC)
    - netperf session on BF2 is way faster (~= 20Gbps)
    - (both w/ LKL and BF2's kernel)
2. BF2 (or DPU) is not powerful enough than x86 hosts
3. satisfy the *ideal* ToE implementation
    - relax CPU/memory usage on host
    - software based implementation (updatable)
    - **but no performance gain**

# To move forward...

- possible performance improvement ?
  - VDPA
  - BF2 kernel instead of LKL
- More powerful, resource-rich DPU
  - BF3?
  - typical x86 machines as an offload devices (not NICs)

# Summary

- kernel offload by **mino**
  - decrease CPU load to NIC
  - copy_{from,to}_user across NIC and host
- transparency
  - application: proper syscall hook
  - kernel/network stack: split but based on the same codebase
- an approach to address *ToE sucks*
  - but no performance gain so far (2023)

# kernel offload with complete host kernel functionalities

Ryo Nakamura (u-tokyo), **Hajime Tazaki (iijlab)**

Linux netdev conference 0x17 (2023)